TRANSMITTAL OF APPEAL BRIEF (Large Entity)

Docket No.
ITL.0149US

In Re Application Of: **Uri Elzur**

#11
LDS
10-8-02

Serial No.	Filing Date	Examiner	Group Art Unit
09/364,085	July 30, 1999	Thu Ha T. Nguyen	2155

Invention: **Associating a Packet With a Flow**

RECEIVED
OCT 03 2002
Technology Center 2100

TO THE ASSISTANT COMMISSIONER FOR PATENTS:

Transmitted herewith in triplicate is the Appeal Brief in this application, with respect to the Notice of Appeal filed on **July 19, 2002**

The fee for filing this Appeal Brief is: **$320.00**

- ☒ A check in the amount of the fee is enclosed.
- ☐ The Commissioner has already been authorized to charge fees in this application to a Deposit Account. A duplicate copy of this sheet is enclosed.
- ☒ The Commissioner is hereby authorized to charge any fees which may be required, or credit any overpayment to Deposit Account No. **20-1504**
  A duplicate copy of this sheet is enclosed.

Signature

Dated: **September 25, 2002**

Fred G. Pruner, Jr., Reg. No. 40,779
Trop, Pruner & Hu, P.C.
8554 Katy Freeway, Suite 100
Houston, Texas 77024
(713) 468-8880 [Phone]
(713) 468-8883 [Fax]

21906
PATENT TRADEMARK OFFICE

I certify that this document and fee is being deposited on 09/25/02 with the U.S. Postal Service as first class mail under 37 C.F.R. 1.8 and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

Signature of Person Mailing Correspondence

Debra Cutrona

Typed or Printed Name of Person Mailing Correspondence

cc:

10/04/2002 SFORD1 00000002 09364085
01 FC:120 320.00 OP

#11

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Uri Elzur	§	Group Art Unit:	2155
		§		
Serial No.:	09/364,085	§		
		§	Examiner:	Thu Ha T. Nguyen
Filed:	July 30, 1999	§		
		§		
Title:	Associating a Packet With a Flow	§ §	Atty. Dkt. No.:	ITL.0149US (P6585)

Board of Patent Appeals & Interferences
Commissioner for Patents
Washington, D.C. 20231

APPEAL BRIEF

Dear Sir:

Applicant hereby appeals from the Final Rejection dated April 19, 2002.

I. REAL PARTY IN INTEREST

The real party in interest is Intel Corporation, the assignee of the present application by virtue of the assignment recorded at Reel/Frame 010329/0486.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

Date of Deposit: September 25, 2002
I hereby certify under 37 CFR 1.8(a) that this correspondence is being deposited with the United States Postal Service as **first class mail** with sufficient postage on the date indicated above and is addressed to the Board of Patent Appeals & Interferences, Commissioner for Patents, Washington, DC 20231.
Debra Cutrona
Debra Cutrona

III. STATUS OF THE CLAIMS

The application was originally filed with claims 1-19. An Amendment (a copy of which is enclosed with this Appeal Brief) is being filed concurrently with this Appeal Brief. The Amendment requests the cancellation of claims 9-13. Because the Amendment narrows down the issues on appeal, it is assumed for purposes of this appeal that claims 9-13 have been canceled. Therefore, claims 1-8 and 14-19 have been finally rejected and are the subject of this appeal.

IV. STATUS OF AMENDMENTS

As set forth above, an Amendment is concurrently being filed with this Appeal Brief, requesting the cancellation of claims 9-13. The cancellation of these claims narrows down the issues on appeal.

V. SUMMARY OF THE INVENTION

Referring to Fig. 4, an embodiment 50 of a computer system in accordance with the invention includes a network controller 52 (a local area network (LAN) controller, for example) that communicates packets of information with other networked computer systems via at least one network wire 53. Unlike conventional network controllers, the network controller 52 is adapted to perform functions that are typically implemented by a processor 54 (a central processing unit (CPU), for example) that executes one or more software layers (a network layer and a transport layer, as examples) of a network protocol stack (a TCP/IP stack, for example). As an example, these functions may include parsing headers of incoming packets to obtain

characteristics (of the packet) that typically are extracted by execution of the software layers. Specification, pp. 4-5.

The characteristics, in turn, may identify an application that is to receive data of the packet. In this context, the term "application" may generally refer to a user of one of the protocol layers (layers 1, 2, 3 or 4, as examples). Due to this identification by the network controller 52, the network controller 52 (and not a software layer of the stack) may directly control the transfer of the packet data to a buffer (in a system memory 56) that is associated with the application. As a result of this arrangement, data transfers between the network controller 52 and the system memory 56 may take less time and more efficiently use memory space, as further described below. Specification, p. 5.

Referring to Fig. 5, the network controller 52 may include hardware, such as a receive path 92, to perform functions to process packets that are received from the network. For example, the receive path 92 may include a receive parser 98 to parse a header of each packet to extract characteristics of the packet, such as characteristics that associate a particular flow with the packet. Because the receive path 92 may be receiving incoming packets from many different flows, the receive path 92 may include a memory 100 that stores entries, called flow tuples 140. Each flow tuple 140 uniquely identifies a flow that is to be parsed by the network controller 52. As further described below, the flows indicated by the flow tuples 140 may be changed by the processor's execution of a driver program 57. Specification, p. 5.

The receive parser 98 may use the stored flow tuples 140 in the following manner. First, the receive parser 98 may interact with the memory 100 to compare parsed information from the

incoming packet with the flow tuples 140 to determine if the incoming flow is one of the flows indicated by the flow tuples 140, i.e., the receive parser 98 determines if a "flow tuple hit," occurs. If a flow tuple hit occurs, the receive parser 98 may parse packets that are associated with the flow, and other circuitry (of the controller 52) may also process the packet based on the detected flow, as further described below. Specification, pp. 5-6.

Referring also to Fig. 6, each flow tuple 140 may include fields that identify characteristics of a particular flow. As an example, in some embodiments, at least one of the flow tuples 140 may be associated with a Transmission Control Protocol (TCP), a User Datagram Protocol (UDP) or a Real-time Transport Protocol, as examples. The flow tuple 140 may include a field 142 that indicates an internet protocol (IP) destination address (i.e., the address of the computer system to receive the packet); a field 144 that indicates an IP source address (i.e., the address of a computer system to transmit the packet); a field 146 that indicates a TCP destination port (i.e., the address of the application that caused generation of the packet); a field 148 that indicates a TCP source port (i.e., the address of the application that is to receive the packet); a field 150 that indicates security/authentication attributes of the packet; and a security parameter index (SPI) field 152 that may be used by the computer system 50 to identify a secure flow. Other flow tuples 140 may be associated with other network protocols, such as a User Datagram Protocol (UDP), for example. Specification, p. 6.

In some embodiments, the receive parser 98 may use a subset of the flow tuple 140 to identify a particular flow. For example, in some embodiments, the receive parser 98 may use the fields 142, 150 and 152 to identify a flow tuple hit. As described further below, the fields 142,

144, 146 and 148 may be used to identify specific types of flow, such as, zero copy flows. Specification, p. 6.

The above references to specific network protocols are intended to be examples only and are not intended to limit the scope of the invention. Additional flow tuples 140 may be stored in the memory 100 and existing flow tuples 140 may be removed from the memory 100 via execution of the driver program 57 by the processor 54. In some embodiments, the memory 100 may also store information fields 141. Each field 141 may be associated with a particular flow tuple 140 and may indicate, for example, a handler that identifies (for the network protocol stack) the flow and a pointer to a buffer of a system memory 56, as further described below. Specification, pp. 6-7.

If the receive parser 98 recognizes (via the flow tuples 140) the flow that is associated with the incoming packet, then the receive path 92 may further process the packet. In some embodiments, the receive parser 98 may indicate (to other circuitry of the network controller 52 and eventually to a network protocol stack) recognition of the flow associated with a particular packet and other detected attributes of the packet. Specification, p. 7.

If the receive parser 98 doesn't recognize the flow, then the receive path 92 passes the incoming packet via a Peripheral Component Interconnect (PCI) interface 130 to software layers of a network protocol stack (a TCP/IP stack, for example) of the computer system 50 for processing. The PCI Specification is available from The PCI Special Interest Group, Portland, Oregon 97214. Other bus interfaces may be used in place of the PCI interface 130 to interface the network controller 52 to buses other than a PCI bus. In some embodiments, the computer

system 50 may execute an operating system that provides at least a portion of some layers (network and transport layers, for example) of the protocol stack. Specification, p. 7.

In some embodiments, even if the receive parser 98 recognizes the flow, additional information may be needed before receive path 92 further processes the incoming packet 52. For example, an authentication/encryption engine 102 may authenticate and/or decrypt the data portion of the incoming packet based on the information that is indicated by the IP security header of the packet. In this manner, if the IP security header indicates that the data portion of the incoming packet is encrypted, then the engine 102 may need a key to decrypt the data portion. Specification, p. 7.

For purposes of providing the key to the engine 102, the network controller 52 may include a key memory 104 that stores different keys that may be indexed by the different associated flows, for example. Additional keys may be stored in the key memory 104 by the processor's execution of the driver program 57, and existing keys may be removed from the key memory 104 by the processor's execution of the driver program 57. In this manner, if the engine 102 determines that the particular decryption key is not stored in the key memory 104, then the engine 102 may submit a request (via the PCI interface 130) to the driver program 57 (see Fig. 4) for the key. In this manner, the driver program 57, when executed by the processor 54, may cause the processor 54 to furnish the key in response to the request and interact with the PCI interface 130 to store the key in the key memory 104. In some embodiments, if the key is unavailable (i.e., the key is not available from the driver program 57 or is not stored in the key memory 104), then the engine 102 does not decrypt the data portion of the packet. Instead, the

PCI interface 130 stores the encrypted data in a predetermined location of the system memory 56 (see Fig. 4) so that software of one or more layers of the protocol stack may be executed to decrypt the data portion of the incoming packet. Specification, pp. 7-8.

After the parsing, the processing of the packet by the network controller 52 may include bypassing the execution of one or more software layers that are associated with the network protocol stack. For example, the receive path 92 may include a zero copy parser 110 that, via the PCI interface 130, may copy data associated with the packet into a memory buffer 304 (see Fig. 7) that is associated with the application layer. In this manner, an application may have one or more associated buffers for receiving the packet data. The operating system creates and maintains the buffers 304 in a virtual address space, and the operating system reserves a multiple number of physical four kilobyte (KB) pages for each buffer 304. The operating system also associates each buffer 304 with a particular application so that the application may use the data stored in the buffer 304. Specification, p. 8.

As described below, to accomplish the direct transfer of packet data from the network controller 52 to the buffers 304, the operating system causes the processor 54 to provide a pointer (to the network controller 52) that points to one of the buffers 304. The indicated buffer 304 may be a buffer allocated by the application for its sole use or a buffer the operating system hands to the network controller 52 to be associated with one of the predefined flows that are to be serviced with zero copy. In the latter case, the operating system will later re-map the buffer to the virtual address space of the application. The zero copy parser 110 uses the flow handle to associate the frame with a zero copy buffer and copy the data directly into that buffer. The

above-described arrangement of transferring data into the buffers 304 is to be contrasted to conventional arrangements that may use intermediate buffers (that are associated with the data link and/or the transport layer) to transfer packet data from the network controller to application layer buffers, as described below. Specification, pp. 8-9.

Referring to Fig. 8, for example, a typical network controller 300 does not directly transfer packet data into the buffers 304 because the typical network controller 300 does not parse the incoming packets to obtain information that identifies the flow or destination application. Instead, the typical network controller 300 transfers the data portion of the packet into packet buffers 302 that are associated with the data link layer. In contrast to the buffers 304, each buffer 302 may have a size range of approximately 1518 bytes (as an example), i.e., the approximate size range of data for a particular packet. The execution of the transport layer (by the processor 54) subsequently associates the data with the appropriate applications and causes the data to be transferred from the buffers 302 to the buffers 304. Specification, p. 9.

Referring back to Fig. 7, in contrast to the conventional arrangement described above, the network controller 52 may use the zero copy parser 110 to bypass the buffers 302 and copy the data portion of the packet directly into the appropriate buffer 304. To accomplish this, the zero copy parser 110 (see Fig. 5) may receive an indication of the TCP destination port (as an example) from the receive parser 98 that, as described above, extracts this information from the header. The TCP (or other layer 4 protocol, e.g., RTP) destination port uniquely identifies the application that is to receive the data and thus, identifies the appropriate buffer 304 for the packet data. Besides transferring the data portions to the buffers 304, the zero copy parser 110 may

handle control issues between the network controller 52 and the protocol stack and may handle cases where an incoming packet is missing, as described below. Specification, p. 9.

The zero copy parser may use a flow context memory 112 to store flow context fields 113 that indicates the particular flows in which zero copying is to be performed. Each context field 113 may be associated with an information field 115 (also stored in the flow context memory 112) that indicates, for example, handles that are associated with the various flows indicated by the flow context fields 113 and other information like addresses, for example. Specification, pp. 9-10.

Referring to Fig. 5, besides the components described above, the receive path 92 may also include one or more first-in-first-out (FIFO) memories 106 to temporarily store the incoming packets through the receive path 92. A checksum engine 108 (of the receive path 92) may be coupled between the FIFO memory(ies) 106 and the PCI interface 130 for purposes of verifying checksums that are embedded in the packets. Specification, p. 10.

The receive path 92 may be interfaced to a PCI bus 72 via the PCI interface 130. The PCI interface 130 may include an emulated direct memory access (DMA) engine 131 that is used for purposes of transferring the data portions of the packets directly into the buffers 304 or 302 (when zero copy is not used). In this manner, the zero copy parser 110 may use one of a predetermined number (sixteen, for example) of DMA channels emulated by the DMA engine 131 to transfer the data into the appropriate buffer 304. In some embodiments, it is possible for each of the channels to be associated with a particular buffer 304. However, in some embodiments, when the protocol stack (instead of the zero copy parser 110) is used to transfer

the data portions of the packets the DMA engine 131 may use a lower number (one, for example) of channels for these transfers. Specification, p. 10.

In some embodiments, the receive path 92 may include additional circuitry, such as a serial-to-parallel conversion circuit 96 that may receive a serial stream of bits from a network interface 90 when a packet is received from the network wire 53. In this manner, the conversion circuit 96 packages the bits into bytes and provides these bytes to the receive parser 98. The network interface 90 may be coupled to generate and receive signals to/from the network wire 53. Specification, p. 10.

In addition to the receive path 92, the network controller 52 may include other hardware circuitry, such as a transmit path 94, to transmit outgoing packets to the network. In the transmit path 94, the network controller 52 may include a transmit parser 114 that is coupled to the PCI interface 130 to receive outgoing packet data from the computer system 50 and form the header on the packets. To accomplish this, in some embodiments, the transmit parser 114 stores the headers of predetermined flows in a header memory 116. Because the headers of a particular flow may indicate a significant amount of the same information (port and IP addresses, for example), the transmit parser 114 may slightly modify the stored header for each outgoing packet and assemble the modified header onto the outgoing packet. As an example, for a particular flow, the transmit parser 114 may retrieve the header from the header memory 116 and parse the header to add such information as sequence and acknowledgment numbers (as examples) to the header of the outgoing packet. A checksum engine 120 may compute

checksums for the IP and network headers of the outgoing packet and incorporate the checksums into the packet. Specification, pp. 10-11.

The transmit path 94 may also include an authentication and encryption engine 126 that may encrypt and/or authenticate the data of the outgoing packets. In this manner, all packets of a particular flow may be encrypted and/or authenticated via a key that is associated with the flow, and the keys for the different flows may be stored in a key memory 124. In some embodiments, new keys may be added to the key memory 124 and existing keys may be modified or deleted by information passed through the transmit path 94 via fields of a control packet. The transmit path 94 may also include one or more FIFO memories 122 to synchronize the flow of the packets through the transmit path 94. A parallel-to-serial conversion circuit 128 may be coupled to the FIFO memory(ies) 122 to retrieve packets that are ready for transmission for purposes of serializing the data of the outgoing packets. Once serialized, the circuit 128 may pass the data to the network interface 90 for transmission to the network wire 53. Specification, p. 11.

In some embodiments, the receive 98 and zero copy 110 parsers may include one or more state machines, counter(s) and timer(s), as examples, to perform the following functions for each incoming packet. In the following, it is assumed that the particular flow being described is a zero copy flow. However, the flow may or may not be a zero copy flow in some embodiments. Referring to Fig. 9, the receive parser 98 may parse (block 200) the header of each incoming packet. From the parsed information, the receive parser 98 may determine if the packet needs authentication or decryption, as indicated in diamond 201. Specification, pp. 11-12.

If authentication or encryption is needed, then the receive parser 98 may use the parsed information from the header to determine (diamond 216) if a flow tuple hit has occurred. If not, the receiver parser 98 transfers control to the zero copy parser 110 that performs end of packet checks, as depicted in block 202. Otherwise, the receive parser 98 determines if the associated key is available in the key memory 104, as depicted in diamond 220. If the key is available, then the receive parser 98 may start authentication and/or decryption of the packet as indicated in block 218 before passing control to the zero copy parser 110 that may perform a zero copy of the packet, as indicated in block 202. If the key is not available, the receive parser 98 may transfer control to the zero copy parser 110 to perform a zero copy operation, as indicated in block 202. Specification, p. 12.

After performing the zero copy operation (block 202), the zero copy parser 110 may perform end of packet checks, as indicated by block 204. In these checks, the receive parser 98 may perform checks that typically are associated with the data link layer. For example, the receive parser 98 may ensure that the packet indicates the correct Ethernet MAC address, no cyclic redundancy check (CRC) errors have occurred, no receive status errors (collision, overrun, minimum/maximum frame length errors, as examples) have occurred and the length of the frame is greater than a minimum number (64, for example) of bytes. The receive parser 98 may perform checks that typically are associated with the network layer. For example, the receive parser 98 may check on the size of the IP packet header, compute a checksum of the IP header, determine if the computed checksum of the IP header is consistent with a checksum indicated by the IP header, ensure that the packet indicates the correct IP destination address and determine if

the IP indicates a recognized network protocol (the TCP or UDP protocols, as examples). The receive parser 98 may also perform checks that are typically associated with functions that are performed by the processor's execution of software that is associated with the transport layer. For example, the receive parser 98 may determine if the size of the protocol header is within predefined limits, may compute a checksum of the protocol header, and may determine if flags called ACK, URG, PSH, RST, FIN and/or SYN flags are set. If the PSH flag is set, then the receiver parser 98 may indicate this event to the driver program. If the RST, FIN or SYN flags are set, the receive parser 98 may surrender control to the transport layer. If the ACK flag is sent, then the receive parser 98 may interact either with the driver program 57 or the transmit path 94 to transmit an acknowledgment packet, as further described below. Specification, pp. 12-13.

After the checks are complete, the zero copy parser 110 may determine (diamond 205) whether a data link layer occurred, an error that may cause the packet to be unusable. If this is the case, then the zero copy parser 110 may reclaim (block 205) the memory that the driver program allocated for the packet, reclaim (block 207) the memory that was allocated for zero copy of the packet and reset (block 209) the DMA channel (emulated by the DMA engine 131) that was associated with the packet. Otherwise, the zero copy parser 110 compiles an error statistics stack for the protocol stack. Specification, p. 13.

Referring to Fig. 10, the zero copy parser 110 may perform the following functions to perform a zero copy operation. First, the zero copy parser 110 may determine the memory address at which to store the data, as further described below. Next, the zero copy parser 110

may determine (diamond 258) whether a packet is missing, and if so, the zero copy parser 110 reserves (block 260) memory space for the missing packet. The zero copy parser 110 subsequently performs (block 262) a zero copy operation to copy the packet into the memory 56. Specification, p. 13.

Next, the zero copy parser 110 may update (block 264) a count of received packets for the flow. The zero copy parser 110 then determines (diamond 266) whether it is time to transmit an acknowledgment packet back to the sender of the packet based on the number of received packets in the flow. In this manner, if the count exceeds a predetermined number, then the receive parser 98 may either (depending on the particular embodiment) notify (block 268) the driver program 57 (see Fig. 4) or notify (block 270) the transmit parser 114 of the need to transmit an acknowledgment packet. Thus, in the latter case, the transmit parser 114 may be adapted to generate an acknowledgment packet, as no data for the data portion may be needed from the application layer. The receive parser 98 transitions from either block 268 or 270 to diamond 200 (see Fig. 9) to check for another received packet. After an acknowledgment packet is transmitted, the receive parser 98 may clear the count of received packets for that particular flow. Specification, pp. 13-14.

A state diagram that illustrates transfer of control from the stack to the network controller 52 in a synchronized manner, per flow, of the receive path 92 is illustrated in Fig. 11. When the software that is associated with the network protocol stack is handling the parsing and processing of the packets, then the receive path 92 remains in an IDLE state. However, in some embodiments, although the transport layer may be executed by the processor 54 to initially

handle the packets, subsequently control of the processing may be transferred to the network controller 52. In this manner, the processor 54, through execution of the driver program 57, may interact with the PCI interface 130 to place the receive path 92 in a MONITOR state. Specification, p. 14.

In the MONITOR state, the receive parser 98 checks the integrity of the incoming packets that are associated with predetermined flows (as indicated by the flow tuples 140) and indicates the results of the check, as described above. For each predetermined flow to be monitored, the memory 100 may store an information field 141 that is associated with the flow. As an example, the information 141 may indicate a handle that indicates the flow to the network stack, a TCP sequence number (as an example) and a pointer to the appropriate network layer buffer 302 (when zero copy is not used). If the receive parser 98 needs a pointer to another buffer 302, then the receive parser 98 may notify the driver program 57 that (in a GET_NEXT_BUFF1 state), in turn, provides the pointer to the next buffer 302, and in response, the receive parser 98 may update the associated field 141. The GET_NEXT_BUFF1 state is related to buffers 304 and is used in the case when zero copy is used. This state machine and this particular state transition may not be used in some embodiments. The stack may also communicate to the network controller 52 to start zero copy from sequence number X or greater than X and from memory address Y that corresponds to that X, thus eliminating this synchronization process. Specification, p. 14.

If the zero copy parser 98 (by using the flow context indications 113) detects that packets from a particular flow are to be zero copied, then the network controller 52 transitions to a ZERO

COPY state. In the ZERO COPY state, the zero copy parser 98 uses the information field 115 that is associated with each zero copy flow to identify such information as the handle that is passed to the network stack (to identify the flow) and the pointer to the appropriate application buffer 304. If a pointer to another buffer 304 is needed, then the zero copy parser 98 requests another pointer from the driver program 57. In response, the driver program 57 (in a GET_NEXT_BUFF2 state) transfers an indication of the pointer to the network controller 52 for use by the zero copy parser 110. In other embodiments it is the responsibility of the application or stack to provide enough buffers for a zero copy flow. In some embodiments, in case the network controller 52 runs out of buffers, the network controller 52 uses the software-based receive procedure. The zero copy parser 110, in response, may update the information field 115. The zero copy parser 110, in response, may update the information field 115. Specification, pp. 14-15.

In some embodiments, the driver program 57 may cause the processor 54 to exit the MONITOR state or ZERO COPY state and return to the IDLE state. The driver program 57 may cause the processor 54 to interact with the PCI interface 131 to add/remove a particular flow context indication 113 to/from the memory 112 and may cause the processor 54 to add/remove a particular flow tuple 140 to/from the flow memory 100. Specification, p. 15.

Referring to Fig. 12, in the performance of the zero copy operation, the zero copy parser may perform the following functions to transfer the packet data directly to the buffers 304. First, the zero copy parser 110 may determine if control of the transfer needs to be synchronized between the zero copy parser 110 and execution of the software that is associated with the layers

(the data link and transport layers, as examples) of the network protocol stack. In this context, the term "synchronization" generally refers to communication between the stack and the zero copy parser 110 for purposes of determining a transition point at which one of the entities (the stack or the zero copy parser 110) takes control from the other and begins transferring data into the buffers 304. Without synchronization, missing packets may not be detected. Therefore, when control passes from the stack to the parser 110 (and vice versa), synchronization may need to occur, as depicted in block 254. Specification, pp. 15-16.

Thus, one scenario where synchronization may be needed is when the zero copy parser 110 initially takes over the function of directly transferring the data portions into the buffers 304. In this manner, if the zero copy parser 110 determines (diamond 250) that the current packet is the first packet being handled by the zero copy parser 110, then the parser 110 synchronizes the packet storage, as depicted by block 254. If not, the zero copy parser 110 determines (diamond 252) if an error has occurred, as described below. For purposes of determining when the transition occurs, the zero copy parser 110 may continually monitor the status of a bit that may be selectively set by the driver program 57, for example. Another scenario where synchronization is needed is when an error occurs when the zero copy parser 110 is copying the packet data into the buffers 304. For example, as a result of the error, the stack may temporarily resume control of the transfer before the zero copy parser 110 regains control. Thus, if the zero copy parser 110 determines (diamond 252) that an error has occurred, the zero copy parser 110 may transition to the block 254. Specification, p. 16.

Synchronization may occur in numerous ways. For example, the zero copy parser 110 may embed a predetermined code into a particular packet status information to indicate to the stack that the zero copy parser 110 handles the transfer of subsequent packets. The stack may do the same. Specification, p. 16.

Occasionally, the incoming packets of a particular flow may be received out of sequence. This may create a problem because the zero copy parser 110 may store the data from sequential packets one after the other in a particular buffer 304. For example, packet number "267" may be received before packet number "266," an event that may cause problems if the data for packet number "267" is stored immediately after the data for packet number "265." To prevent this scenario from occurring, in some embodiments, the zero copy parser 110 may reserve a region 308 (see Fig. 7) in the particular buffer 304 for the missing packet data, as indicated in block 260 (Fig. 11). For purposes of determining the size of the missing packet (and thus, the amount of memory space to reserve), the zero copy parser 110 may use the sequence numbers that are indicated by the adjacent packets in the sequence. In this manner, the sequence number indicates the byte number of the next successive packet. Thus, for the example described above, the acknowledgment numbers indicated by the packet numbers "265" and "267" may be used to determine the boundaries of the region 308. Specification, pp. 16-17.

The zero copy parser 110 subsequently interacts with the PCI interface 130 to set up the appropriate DMA channel to perform a zero copy (step 262) of the packet data into the appropriate buffer 304. The zero copy parser 110 determines the appropriate buffer 304 via the destination port that is provided by the receive parser 98. Specification, p. 17.

Referring back to Fig. 4, besides the network controller 52, the computer system 50 may include a processor 54 that is coupled to a host bus 58. In this context, the term "processor" may generally refer to one or more central processing units (CPUs), microcontrollers or microprocessors (an X86 microprocessor, a Pentium microprocessor or an Advanced RISC Controller (ARM), as examples), as just a few examples. Furthermore, the phrase "computer system" may refer to any type of processor-based system that may include a desktop computer, a laptop computer, an appliance or a set-top box, as just a few examples. Thus, the invention is not intended to be limited to the illustrated computer system 50 but rather, the computer system 50 is an example of one of many embodiments of the invention. Specification, p. 17.

The host bus 58 may be coupled by a bridge, or memory hub 60, to an Accelerated Graphics Port (AGP) bus 62. The AGP is described in detail in the Accelerated Graphics Port Interface Specification, Revision 1.0, published in July 31, 1996, by Intel Corporation of Santa Clara, California. The AGP bus 62 may be coupled to, for example, a video controller 64 that controls a display 65. The memory hub 60 may also couple the AGP bus 62 and the host bus 58 to a memory bus 61. The memory bus 61, in turn, may be coupled to a system memory 56 that may, as examples, store the buffers 304 and a copy of the driver program 57. Specification, p. 17.

The memory hub 60 may also be coupled (via a hub link 66) to another bridge, or input/output (I/O) hub 68, that is coupled to an I/O expansion bus 70 and the PCI bus 72. The I/O hub 68 may also be coupled to, as examples, a CD-ROM drive 82 and a hard disk drive 84. The I/O expansion bus 70 may be coupled to an I/O controller 74 that controls operation of a

floppy disk drive 76 and receives input data from a keyboard 78 and a mouse 80, as examples. Specification, pp. 17-18.

VI. ISSUES

A. Can claims 1 and 3-8 be anticipated and claim 2 be rendered obvious in view of a reference that fails to teach storing a table in a memory of a peripheral?

B. Can claims 14, 15 and 17-19 be anticipated and claim 16 be rendered obvious in view of a reference that fails to teach a peripheral that includes a peripheral memory that is adapted to store a table that includes entries that identify different packet flows?

VII. GROUPING OF THE CLAIMS

Claims 1-8 can be grouped together; and claims 14-19 can be grouped together.

VIII. ARGUMENT

All claims should be allowed over the cited references for the reasons set forth below.

A. Can claims 1 and 3-8 be anticipated and claim 2 be rendered obvious in view of a reference that fails to teach storing a table in a memory of a peripheral?

The method of independent claim 1 is for use with a computer system and includes storing a table in a memory of a peripheral. The table includes entries that identify different packet flows. The method includes receiving a packet and using the table to associate the packet with one of the packet flows.

The Examiner rejects independent claim 1 under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,141,686 (herein called "Jackowski"). Jackowski generally teaches a plugin application to provide network services. Figs. 4 and 5 generally show a network architecture for

software in the context of an application-classifier plugin. *See, for example*, Jackowski, 6:17-21. Thus, Figs. 4 and 5 generally depict software architectures.

Contrary to the Examiner's contentions, Jackowski neither teaches nor suggests storing a table in a memory of a peripheral. In this manner, the Examiner contends that storing a table in a memory of a peripheral is taught by Jackowski's brief discussion of an "examiner," software that Jackowski describes as collecting statistical information about a network event, as set forth in the Abstract of Jackowski beginning on line 61 of column 4 and extending through line 11 of column 5. Final Office Action, p. 4. However, the cited language neither teaches nor suggests storing a table in a memory of a peripheral.

Referring to the Detailed Description of Jackowski in Fig. 5 and the associated text, Jackowski describes an application-classifier engine (ACE) plugin. *See, for example*, Jackowski, 7:34-54. This plugin includes an application classifier plugin 51 and other software components that are executed by a service provider 50, the main computing unit of the system that is depicted in Fig. 5. Jackowski, 8:7-56. As part of these software components (*See* heading at Jackowski, 8:58) of the plugin, Jackowski describes an ACE consolidator 60 and states that the ACE consolidator stores tables. Jackowski, 9:15-17. The consolidator 60 is a software module, i.e., software that causes the service provider 50 to store data for the tables in apparently *some* memory. However, Jackowski is not specific as to the identity of the physical memory in which the tables are stored. For example, the tables may be stored in the working memory of a computer system, not in a memory of a peripheral.

Thus, the Examiner is effectively reading the "peripheral" limitation out of claim 1, as claim 1 specifically sets forth storing tables in a memory of a peripheral, not just any memory. Words in a claim are generally given their ordinary and customary meaning that would be given by persons experienced in the field of the invention, unless it is apparent from the patent and the prosecution history that the inventor intended the words to have a different meaning. *Hoechst Celanese Corp. v. BT Chemicals Ltd.*, 38 USPQ2d 1126, 1129 (Fed. Cir. 1996), *cert, denied*, 117 S. Ct. 275 (1996).

The term "peripheral" has a specific meaning in the art. As evidence of the meaning of "peripheral," attached to this Appeal Brief as Appendix B is the definition of "peripheral" from "FOLDOC," an online dictionary. As can be appreciated from this definition, the term "peripheral" is not an arbitrary term, but rather, a "peripheral" is a part of a computer that does not include the CPU or working memory of the computer. Thus, "a memory of a peripheral" is not taught by a general disclosure or implication of the presence of a memory.

Therefore, Jackowski does not teach storing a table in a memory of a peripheral. Claims 2-8 are patentable for at least the reason that these claims depend from an allowable claim.

Thus, the rejections of claims 1-8 are improper and should be reversed.

B. Can claims 14, 15 and 17-19 be anticipated and claim 16 be rendered obvious in view of a reference that fails to teach a peripheral that includes a peripheral memory that is adapted to store a table that includes entries that identify different packet flows?

The computer system of independent claim 14 includes a system memory, a processor and a peripheral. The peripheral includes a peripheral memory, a first interface, a second

interface and a circuit. The peripheral memory is adapted to store a table that includes entries that identify different packet flows. The first interface is adapted to receive a packet, and the second interface is adapted to communicate with the system memory. A circuit of the peripheral is adapted to use the table to associate the packet with one of the packet flows and based on the association, interact with the second interface to selectively transfer a portion of the packet to the system memory for processing by the processor.

The Examiner rejects independent claim 14 under 35 U.S.C. § 102(e) in view of Jackowski. The computer system of claim 14 includes a processor and a peripheral. The peripheral includes a peripheral memory that is adapted to store a table that includes entries that identify different packet flows. The language of claim 14 distinguishes two types of memories: a system memory and a peripheral memory of a peripheral. As discussed above in connection with Issue A, Jackowski states that the ACE consolidator 60 stores tables, thereby implying the storage of these tables in a memory. However, Jackowski is not specific as to the identity of this memory. The term "peripheral" has a specific meaning in the art not considered by the Examiner. Therefore, Jackowski does not teach a peripheral that includes a peripheral memory that stores a table that includes entries that identify different packet flows.

Thus, the rejections of claims 14-19 are improper and should be reversed.

IX. CONCLUSION

The Assignee requests that each of the final rejections be reversed and that the claims subject to this appeal be allowed to issue.

Respectfully submitted,

Date: September 25, 2002

Fred G. Pruner, Jr., Reg. No. 40,779
TROP, PRUNER & HU, P.C.
8554 Katy Freeway, Suite 100
Houston, TX 77024-1805
713/468-8880 [Phone]
713/468-8883 [Facsimile]

21906
PATENT TRADEMARK OFFICE

APPENDIX A - CLAIMS

The claims on appeal are:

1. A method for use with a computer system, comprising:

storing a table in a memory of a peripheral, the table including entries identifying different packet flows;

receiving a packet; and

using the table to associate the packet with one of the packet flows.

2. The method of claim 1, wherein the packet indicates a header and the act of using the table comprises:

parsing the packet to identify at least one characteristic of the packet; and

comparing said at least one characteristic to the entries.

3. The method of claim 1, wherein said at least one characteristic comprises:

a port number being associated with an application.

4. The method of claim 1, wherein said at least one characteristic comprises:

a security attribute.

5. The method of claim 1, further comprising:

based on the association, selectively using hardware to process the packet.

6. The method of claim 1, further comprising:

based on the association, selectively executing software to process the packet.

7. The method of claim 1, wherein the peripheral comprises:

a network controller.

8. The method of claim 1, further comprising:

storing the packet in another memory of the peripheral.

14. A computer system comprising:

a system memory;

a processor; and

a peripheral comprising:

a peripheral memory adapted to store a table including entries identifying different packet flows;

a first interface adapted to receive a packet;

a second interface adapted to communicate with the system memory; and

a circuit adapted to:

use the table to associate the packet with one of the packet flows, and

based on the association, interact with the second interface to selectively transfer a portion of the packet to the system memory for processing by the processor.

15. The computer system of claim 14, wherein the peripheral comprises:

a network controller.

16. The computer system of claim 14, wherein the packet indicates a header and the circuit is further adapted to:

parse the packet to identify flow characteristics of the packet; and

compare the characteristics to the entries to associate the packet with one of the packet flows.

17. The computer system of claim 14, wherein the characteristics comprise:

a port number.

18. The computer system of claim 14, wherein the characteristic comprise:

a security attribute.

19. The computer system of claim 14, further comprising:

another memory coupled to the first interface and adapted to store the packet.

APPENDIX B

peripheral

<*hardware*> (Or "peripheral device", "device") Any part of a computer other than the CPU or working memory, i.e. disks, keyboards, monitors, mice, printers, scanners, tape drives, microphones, speakers, cameras, to list just the less exotic ones.

High speed working memory, such as RAM, ROM or, in the old days, core would not normally be referred to as peripherals. The more modern term "device" is also more general in that it is used for things such as a pseudo-tty, a RAM drive or a network adaptor.

(1999-02-10)

Try this search on OneLook / Google

Nearby terms: perfect programmer syndrome « PERFORM « periodic group « **peripheral** » Peripheral Component Interconnect » peripheral device » Peripheral Technology Group